Sketch-based Instancing of Parameterized 3D Models

by

Dan Xiao

M.Sc., Zhejiang University, 2002

A THESIS SUBMITTED IN PARTIAL FULFILLMENT OF

THE REQUIREMENTS FOR THE DEGREE OF

Masters of Science

in

THE FACULTY OF GRADUATE STUDIES

(Department of Computer Science)

We accept this thesis as conforming
to the required standard

The University of British Columbia

September 2004

Library Authorization

In presenting this thesis in partial fulfillment of the requirements for an advanced degree at the University of British Columbia, I agree that the Library shall make it freely available for reference and study. I further agree that permission for extensive copying of this thesis for scholarly purposes may be granted by the head of my department or by his or her representatives. It is understood that copying or publication of this thesis for financial gain shall not be allowed without my written permission.

DAN XIAO
Name of Author *(please print)*

04/10/2004
Date (dd/mm/yyyy)

Title of Thesis: Sketch-based Instancing of Parameterized 3D Models

Degree: Master Year: 2

Department of Computer Science
The University of British Columbia
Vancouver, BC Canada

Abstract

We present a system for constructing 3D models from simple hand-drawn sketches. The system exploits a priori knowledge about the type of object being sketched. In particular, we assume that the class of object being drawn is known, (e.g., a rocket ship) and that the object will be drawn using a fixed number of known components. By using these assumptions, we show that we can build a sketch-based modeling system that, while object specific, is capable of quickly creating geometric models that are beyond the scope of current sketch-based modeling approaches. Key to our approach is the use of a K-means classifier for labeling each drawn stroke as being a particular component in the generic model. We demonstrate our approach by applying this classifier to face and rocket sketches.

Contents

List of Tables

List of Figures

Acknowledgements

I would like to give my great gratitude to my supervisor, Dr. Michiel van de Panne, for his invaluble advice on my thesis work. Without him, this would never have been completed. I am also very grateful to Dr. Nando de Freitas, who is giving the interesting machine learning course and stimulate the idea in this thesis.

To Jason Harrison, who contribute useful books and papers. Also to Peng Zhao, Yizhen Cai and Xiaojing Wu for providing several discussions.

DAN XIAO

The University of British Columbia
September 2004

Chapter 1

Introduction

1.1 Motivation

"A picture speaks a thousand words". People often rely on sketches when they try to convey ideas to others. We all have the experience of drawing something. Of all drawers, children are among the most fanatic. They like to convert what they see and think into pictures. Sometimes their pictures are simply drawn but they are at the same time capable of conveying much information. Creating and animating 3D models is currently not possible, however, without significant training and expertise.

New technologies are thriving in our modern society. One common characteristic of successful technologies is their ease of use, so that people can grasp those technologies in a short period of time and use them in an easy way. With devices such as PDAs and Tablet PCs, there exists hardware that largely replicates the feel and function of a pencil or pen or paper. What now remains is for new software applications to exploit these new capabilities. Our system can interpret a limited class of 2D sketeches as 3D objects, and thus represents such a new application.

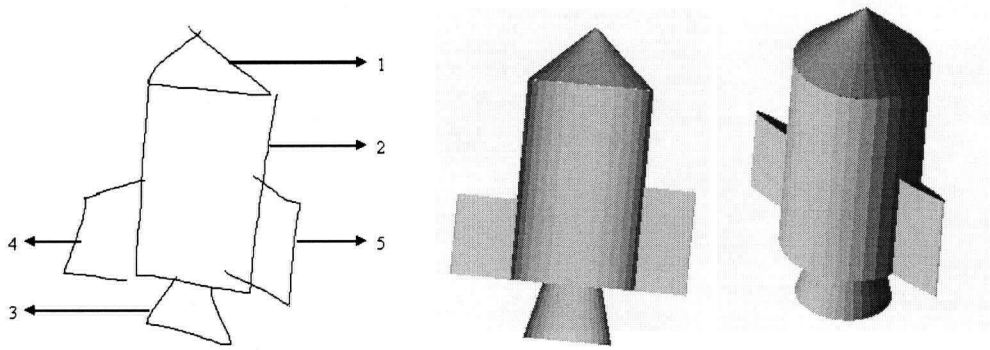

Figure 1.1: An example of sketch recognition and modeling.

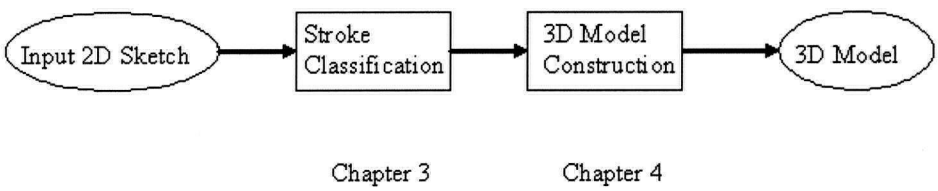

Figure 1.2: System Overview.

Stroke	Resulting component
1	cone
2	body
3	tail
4	left wing
5	right wing

Table 1.1: Correspondences between strokes and components of a rocket.

1.2 System Overview

Figure 1.1 shows an example drawn and recognized by our system. Each pen stroke in the graph is recognized as a component of the object and then constructs a component of the 3D model. In the above graph, a rocket is drawn with 5 pen strokes and later, a 3D rocket model is displayed with 5 components. Each pen stroke represents one component and their correspondences are illustrated in Table 1.1. The overall system structure is given in Figure 1.2.

1.3 Application of Sketch Recognition

The application domains of Freehand Sketch Recognition System (FSRS) include:

- 3D modeling based on sketching interface, such as computer aided design (CAD).
- Query by sketch : image database retrieval.

A long-standing dream in computer-aided design is the possibility of using freehand sketching as the language for interactive design. The ability to sketch a 3D object, predict its performance, and re-design it interactively based on physics-based feedback would bring the power of state-of-the-art CAD tools into the critical, early design phase. The enormous potential of sketch-based interfaces is widely recognized, and has been broadly pursued. However, the practical use of such attempts has remained limited because these interfaces have been primarily 2D, losing much of the benefit of mainstream 3D CAD. In order to become truly 3D, a sketch interface must automatically be able to reconstruct the spatial geometry from a single 2D sketch in near real-time, much like a human observer does implicitly.

1.4 Thesis Outline

The rest of this thesis is organized as follows. In Chapter 2, we discuss work related to this project. Chapter 3 explores how to label individual strokes using classification and compares different classifiers. Chapter 4 describes our implemented system. Chapter 5 presents results. We make conclusions and discuss future work in Chapter 6.

Chapter 2

Related Work

2.1 Attributes of a Sketch Recognition System

Freehand sketching can play an important role in the realm of user-interfaces for graphics systems. Sketching appears to be a natural communication language, enabling fast conveyance of qualitative information while not burdening the creativity of the user or disrupting the flow of ideas. As illustrated in Figure 2.1, sketching is a key element in modeling, developing and defining a design result. During the whole design process, sketching often provides the input for conceptual design. [17] suggests that a sketch recognition technology must meet three main requirements: it must deal reliably with the pervasive variability of hand sketches, provide interactive performance, and be easily extensible to new configurations.

User interfaces based on sketching can generally be divided into three categories, according to the level of information they intend to gather from the input sketch [22]:

1. Drawing Pads. Basic sketching is allowed for general purpose drawings. Input strokes are smoothed and many other graphic tools are provided.

2. 2D sketch applications. These applications smooth sketch strokes and classify them into 2D primitives, such as lines, arcs, and splines. Some geometric

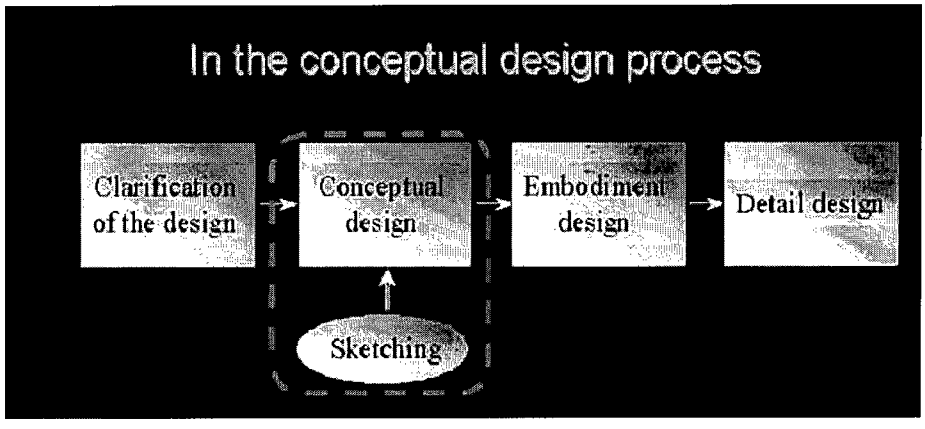

Figure 2.1: Design process [13].

constraints and relationships among the entities are utilized to further refine the sketch.

3. 3D sketchers. Sketches are analyzed as representing rough projections of 3D scenes. Sketch strokes are identified as basic geometrical shapes, such as lines, arcs and corners. However, some of the sketch strokes do not necessarily represent what they appear to be since the analyzed sketch represents a rough projection of a three dimensional scene [11, 14].

2.2 Algorithms for Object Recognition

The sketch recognition problem is strongly related to the object recognition problem in computer vision. A broad class of such object recognition problems have benefited from statistical learning machinery. The use of Principle Component Analysis (PCA) has produced good results [25] in the area of face recognition. On more general object recognition tasks, several other learning methods such as Bayes classifier [21] and decision tree learning [28] have been applied. The technique of boosting is proven to be a viable method of feature selection in [26]. Support vector machine methods have demonstrated success in template matching problems such as

recognizing pedestrians in [16].

[2] and [31] present a novel approach to measure similarity between 2D shapes and exploit it for object recognition. In order to solve the correspondences problem, they attach a descriptor, the shape context, to each point. The shape context at a reference point captures the distribution of the remaining points relative to it, thus offering a globally discriminative characterization. In this way, they treat recognition in a nearest-neighbor classification framework as the problem of finding the stored prototype shape that is maximally similar to that in the image.

2.3 2D Sketch Recognition

Research on sketch recognition and interpretation has a long history. A variety of systems have incorporated gesture recognition into their user interface. The development of a Freehand Sketch Recognition System involves three stages:

1. Stroke-level recognition which interprets the pixels and produces low-level geometric primitives such as lines and curves.

2. 3D modeling based on 2D sketches.

3. Multi-stroke understanding.

Each sketch of an object consists of multiple strokes. The first step in intepreting a sketch is to classify individual pen strokes by processing them. Usually there are two assumptions in this area, one is that each pen stroke symbolizes a single shape, such as a single curve segment or triangle segment, whichever fits the stroke best. Much of the previous work has relied either on using stroke methods in which an entire symbol must be drawn as single stroke [20, 10, 4], or single primitive methods in which each stroke must be a single line, arc, or curve [18, 6, 27]. Another one used in [3] is that multiple primitives can be drawn in the same pen stroke. In this way, a square can be drawn as four individual pen strokes or a single pen stroke

with three 90° bends. The key challenge is determing which bumps and bends are drawn intentionally and which are unintentionally. Arvo and Novins described a kind of stroke-level recognition algorithm [1]. Different from traditional ways of recognizing sketches, their method continuously morphs raw input strokes into ideal geometric shapes.

Another stroke-level recognition algorithm is a domain-independent system for sketch recognition [29]. Users are allowed to draw sketches as naturally as how they do on paper. The system then recognizes the drawing through imprecise stroke approximation which is implemented in a unified and incremental procedure. This method can handle smooth curves and hybrid shapes as gracefully as it does to polylines. With a feature-area verification mechanism and the intelligent adjustment in a post-process, the system can produce the results intended by the user.

Rubine presented a trainable gesture recognizer which used a "linear machine" classifier to discriminate gestures [20]. Each gesture class is associated with a linear evaluation function using 13 features. In order for training, appropriate weights are learned for each attribute in the linear function. The attributes consider aggregate properties of a pen stroke, and it is possible that two different gestures would have the same aggregate properties.

Recent studies show us some interesting gesture-based interfaces in which the user specifies commands by simple drawings. GRANDMA, a tool for building gesture-based applications introduced in [19], supports the integration of gestures and direct manipulation. It allows views that respond to gestures and views that respond to clicks and drags to coexist in the same interface. Our work is closely related to this in that we also a set of stroke features and then apply a classifier to discriminate different gestures.

Landay and Myers advanced an interactive sketching tool called SILK in which designers can quickly sketch out a user interface and transform it into a fully operational system [12]. As the designer sketches, the recognizer of SILK matches

the pen strokes to symbols representing various user interface components, and returns the most likely interpretation. Their recognizer is limited to single-stroke shapes drawn in certain preferred orientations. [23, 24] have developed a program called SketchIT that can transform a sketch of a mechanical device into working designs.

Electronic Cocktail Napkin [5] employs a trainable recognizer that works for multi-stroke shapes. The recognition process consists of glyph (low-level) and configuration (high-level) recognition. A glyph is described by a state transition model of the pen path, the aspect ratio and size of the bounding box, and the number of corner points. Configuration recognition takes the spatial relationships between the glyphs into consideration. This method is sensitive to changes in orientation, and the 3x3 grid may be inadequate for symbols containing small features.

2.4 3D Sketch-based Modeling

Zeleznik et al. have extended the use of gesture recognition for 3D modeling[30]. The SKETCH system they proposed attempts to combine 2D image with 3D computer modeling systems to create an environment for rapidly conceptualizing and editing approximate 3D scenes. All interaction with SKETCH is via a three-button mouse, occasional use of one modifier key on the keyboard, and a single orthographic window onto the 3D scene.

The Teddy system [8] in Figure 2.2 is aimed especially at the modeling of simple free form objects. The system is able to inflate 2D closed curves into 3D objects, which can then be edited using simple gestures. The interface shown in Figure 2.3 is Chateau [7], a prototype systemin which the user gives the hints to the system by highlighting related lines and the system suggests possible operations based on the hints, shown in the thumbnails at bottom.

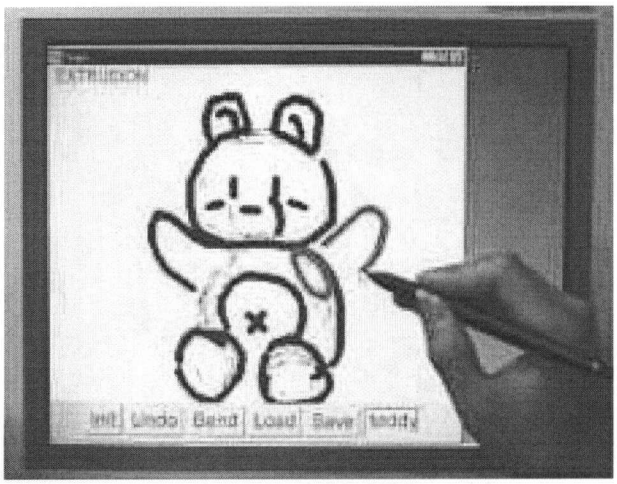

Figure 2.2: Example of user interface: Teddy.

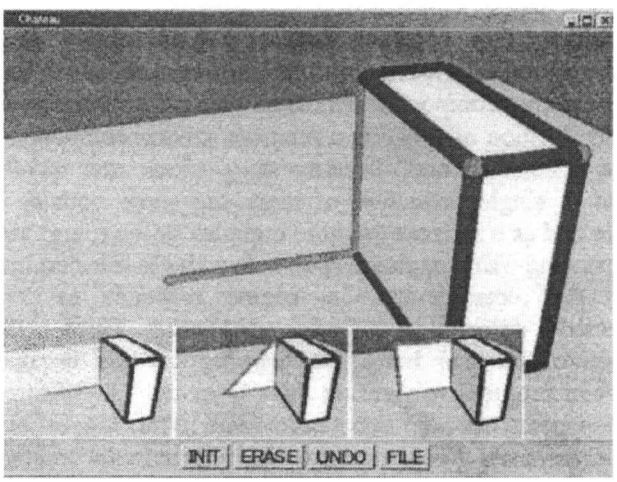

Figure 2.3: Another example of user interface: Chateau.

2.5 Summary

In this chapter, we presented a review of sketch recognition systems, 2D sketch recognition methods, along with some 3D sketch-based modeling examples. In the next chapter, we will present some machine learning methods used for gesture recognition.

Chapter 3

Single-stroke Classification

One of the goals of our work is to obtain an efficent classifer as well as a rich and meaningful feature vectors. In this chapter, we compare several machine learning methods that classify strokes into one of several categories based upon the stroke feature vectors [20]. The selection of useful features and a good classifier will determine the effectiveness of our gesture recognizer.

3.1 Problem

Each gesture is defined as an array g of coordinate values and time.

$$g_p = (x_p, y_p, t_p) \qquad 0 \leq p < P$$

The gesture recognition problem is stated as follows. There is a gesture set with C classes. Given a gesture, we need to determine to which class it belongs. This is done by first extracting a limited set of features. Features are extracted from the gesture g and are used as the input for classification. The feature vector $f = [f_1 \ldots f_P]$ is taken as the training data for classification.

3.2 The Features

As an example, a stroke can be characterized by a set of 11 geometric and 2 dynamic features [20]. This is the set of features we shall use for testing several classification algorithms. Note that many other feasible feature sets could also have been used, with possible varying results. Figure 3.1 shows how to extract these specific features from a stroke. The discription of features are:

- Feature 1 (f_1) : the cosine of the initial angle of the gesture.
- Feature 2 (f_2) : the sine value of the initial angle of the gesture.
- Feature 3 (f_3) : the length of the bounding box diagonal.
- Feature 4 (f_4) : the angle of the bounding box diagonal.
- Feature 5 (f_5) : the distance between the first and the last point.
- Feature 6 (f_6) : the cosine of the angle between the first and last point.
- Feature 7 (f_7) : the sine of the angle between the first and last point.
- Feature 8 (f_8) : the total gesture length.
- Feature 9 (f_9) : the total angle traversed.
- Feature 10 (f_{10}) : the sum of the absolute value of the angle at each mouse point.
- Feature 11 (f_{11}) : the sum of the squared value of those angles.
- Feature 12 (f_{12}) : the maximum speed (squared) of the gesture.
- Feature 13 (f_{13}) : the duration of the gesture.

$$f_1 = \cos\alpha = x_2 - x_0/\sqrt{(x_2 - x_0)^2 + (y_2 - y_0)^2}$$

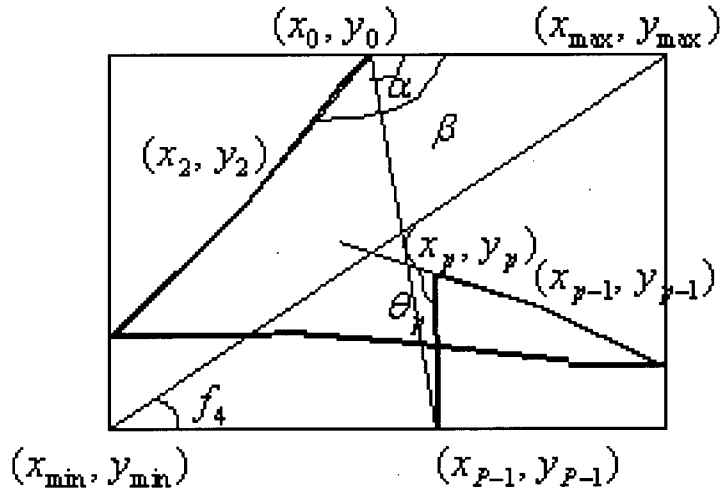

Figure 3.1: Features extracted from a gesture for identification.

$$f_2 = \sin\alpha = y_2 - y_0/\sqrt{(x_2 - x_0)^2 + (y_2 - y_0)^2}$$

$$f_3 = \sqrt{(x_{max} - x_{min})^2 + (x_{max} - x_{min})^2}$$

$$f_4 = \arctan\frac{y_{max} - y_{min}}{x_{max} - x_{min}}$$

$$f_5 = \sqrt{(x_{P-1} - x_0)^2 + (y_{P-1} - y_0)^2}$$

$$f_6 = \cos\beta = (x_{P-1} - x_0)/f_5$$

$$f_7 = \sin\beta = (y_{P-1} - y_0)/f_5$$

Let $\Delta x_p = x_{p+1} - x_p$, $\Delta y_p = y_{p+1} - y_p$

$$f_8 = \sum_{p=0}^{P-2}\sqrt{\Delta x_p^2 + \Delta y_p^2}$$

Let $\theta_p = \arctan\frac{\Delta x_p \Delta y_{p-1} - \Delta x_{p-1}\Delta y_p}{\Delta x_p \Delta x_{p-1} - \Delta y_p \Delta y_{p-1}}$

$$f_9 = \sum_{p=1}^{P-2}\theta_p$$

$$f_{10} = \sum_{p=1}^{P-2} |\theta_p|$$

$$f_{11} = \sum_{p=1}^{P-2} \theta_p^2$$

Let $\Delta t_p = t_{p+1} - t_p$

$$f_{12} = \max_{p=0}^{P-2} \frac{(x_{max} - x_{min})^2 + (x_{max} - x_{min})^2}{\Delta t_p^2}$$

$$f_{13} = t_{P+1} - t_0$$

3.3 Methods for Stroke Classification

We now introduce several possible classification methods with the purpose of determining which one works best. Three machine learning classification schemes are presented in this section: least squares, k-means, expectation-maximisation. The least squares method is a supervised learning technique while the latter two use unsupervised learning.

3.3.1 Least Squares Method

[20] uses a linear machine algorithm for strokes classification, which is a kind of least square approach.

Given C as the number of total classes and F is the number of features, each gesture class c has parameters ω_{ci} for $0 \leq i \leq F$. The evaluations, υ_c, are calculated as follows:

$$\upsilon_c = \omega_{c0} + \sum_{i=1}^{F} \omega_{ci} f_i \qquad 0 \leq c < C$$

The class of a gesture g is the c which maximizes υ_c.

During the training process, we need to determine the weights ω_{ci} from the example gestures. Let f_{cei} be the feature of the example of gesture class c, $0 \leq e <$

E_c, where E_c is the number of training examples of class c. The sample estimate of the mean feature vector per class $\overline{f}_c$ is defined as:

$$\overline{f}_{ci} = \frac{1}{E_c} \sum_{e=0}^{E_c-1} f_{cei}$$

The sample estimate of the covariance matrix of class c, $\sum cij$, is computed as:

$$\Sigma cij = \sum_{e=0}^{E_c-1} (f_{cei} - \overline{f}_{ci})(f_{cei} - \overline{f}_{ci})$$

The $\sum cij$ are averaged to yield $\sum ij$, an estimate of the common covariance matrix.

$$\Sigma ij = \frac{\sum_{c=0}^{C-1} \Sigma cij}{-C + \sum_{c=0}^{C-1} E_c}$$

We can invert the common covariance matrix and use it to get the weights ω_{ci} as follows [20]:

$$\omega_{cj} = \sum_{i=1}^{F} (\Sigma^{-1})_{ij} \overline{f}_{ci} \qquad 1 \leq j \leq F$$

$$\omega_{c0} = -\frac{1}{2} \sum_{i=1}^{F} \omega_{ci} \overline{f}_{ci}$$

3.3.2 K-means Method

We use K-means clustering introduced in [15] for classification. In K-means clustering, K, needs to be determined at the onset. The goal is to divide the objects into K clusters such that some metric relative to the centroids of the clusters is minimized. Various metrics related to the centroids can be minimized, including:

- The maximum distance to its centroid for any object.
- The sum of the average distance to the centroids over all clusters.
- The sum of the variance over all clusters.
- The total distance between all objects and their centroids.

The metric to minimize and the choice of a distance measure will determine the shape of the optimium clusters.

Two different algorithms are available to search for the optimum set of clusters. In the first procedure, the objects are randomly assigned to one of the K clusters. Once this is done, the position of the K centroids are determined, as is the value of the metric to minimize. A global optimization method is then used to reassign some of the objects to different clusters. New centroids are determined, as is the metric to minimize. This procedure is continued until the optimum assignment of objects to clusters is found.

In the second procedure for K-means clustering, placement of the K centroids can be done by the following procedure. Place K points into the space represented by the objects that are being clustered. These points represent initial group centroids.

1. Assign each object to the group that has the closest centroid.
2. When all objects have been assigned, recalculate the positions of the K centroids.
3. Repeat Steps 2 and 3 until the centroids no longer move. This produces a separation of the objects into groups from which the metric to be minimized can be calculated.

A global optimization method is then used to move the position of one or more of the centroids. The above procedure is repeated and new metric value is determined. The object is to move the centroids into a position such that an optimum separation of objects into groups occurrs.

For K-Means clustering, it is necessary to calculate a "distance" between either two objects (one of which may be a cluster seed point) or an object and a group centroid. In this discussion, it is assumed that each object is described by an array of real-valued metrics.

Since we need to minimize the largest or average distance, or the variance, during clustering, it is important that each metric contribute equally to the total distance. In other words, if one metric spans the range [0.0,0.5] and another spans [0.0,100.0], the maximum deviation in the first would have little effect on the total distance, while even a modest separation in the second would have a much larger effect. To remove this dependency on the range spanned by each metric, it is important to first standardize the values. This means that each metric, when compared over the full set of objects should have a mean of 0.0 and a variance (or standard deviation) of 1.0. For each metric, the following steps should be taken to standardize each metric describing the objects.

1. Sum the values of the metric over all objects and divide the sum by the number of objects.
2. Subtract this average value from the metric in all objects.
3. Sum the square of these new values over all objects, divide the sum by the total number of objects, and take its square-root. This is the standard deviation of the new values.
4. Divide the metric by the standard deviation in each object.

3.3.3 Expectation-Maximisation Method

This method supposes that the stroke features have normal Gaussian distributions. The EM method can be applied to build a gesture classifier. The detailed steps are given by Jordan [9]:

1. Initialise.
2. E Step: At iteration t, compute the expectation of the indicators for each i and c:

$$\zeta_{ic} = \frac{p(c)N(x_i|\mu_c, \Sigma c)}{\Sigma_{c'}^{k} p(c')N(x_i|\mu_{c'}, \Sigma_{c'})}$$

and normalize it.

3. M Step: Update the parameters $P(c)$, μ_c, Σ_c.

$$\mu_c = \frac{\sum_{i=1}^{n} \zeta_{ic} x_i}{\sum_{i=1}^{n} n\zeta_{ic}}$$

$$\sum_c = \frac{\sum_{i=1}^{n} \zeta_{ic}(x_i - \mu_c)(x_i - \mu_c)'}{\sum_{i=1}^{n} \zeta_{ic}}$$

$$p(c) = \frac{1}{n} \sum_{i=1}^{n} \zeta_{ic}$$

The EM method is used to find the parameters for the normal distribution of each class and then apply this distribution to classify new, unseen feature vectors. The distribution with the highest probability is defined to be the class to which the gesture belongs.

3.4 Classifier Comparison

The gestures used for evaluation of the three classification algorithms are shown in Figure 3.2. In this figure, four different gesture styles of each class are given.

Performance is evaluated on 3 gesture classes. Figure 3.3 shows the results using the algorithm presented by [20]. The plot demonstrates the recognition rate as a function of the number of training examples per class for evaluation gestures. One line is for 2 classes and the other one is for 3 classes. Figure 3.4 illustrates the results with k-means. The results when applying EM are shown in Figure 3.5.

From the above three figures, we can see that the K-means method gives the best performance. For the K-means method, in the cases where 3 gesture classes are recognized by a classifier trained with 10 or more examples per class, at least 96% of the test gestures are classified correctly. When there are only 2 gesture classes, the classifier trained with 10 or more examples per class yields 100% correct classification.

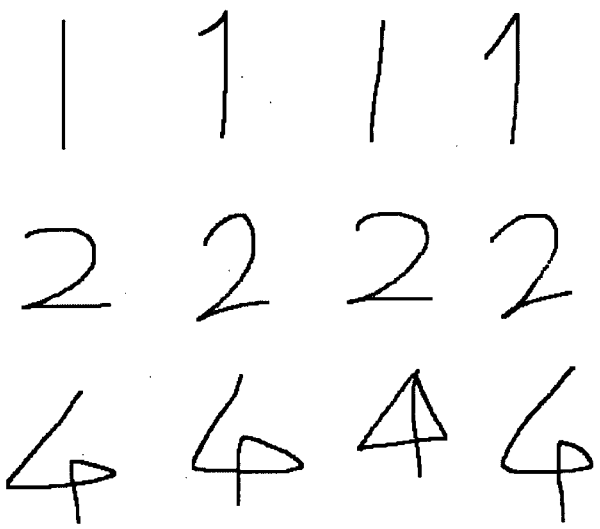

Figure 3.2: Examples from the gesture set used for evaluation.

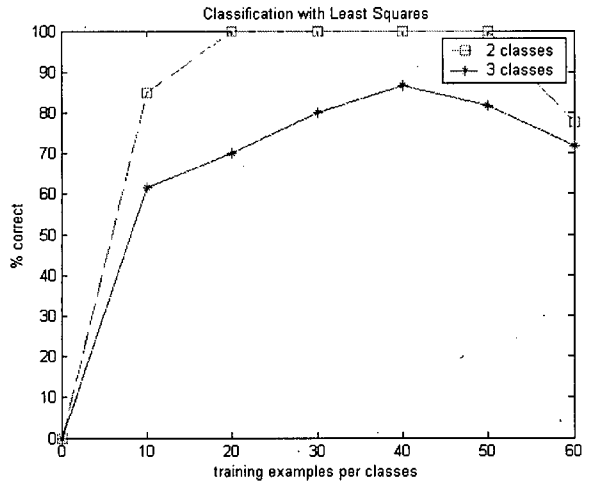

Figure 3.3: Recognition rate vs. training size for least square method.

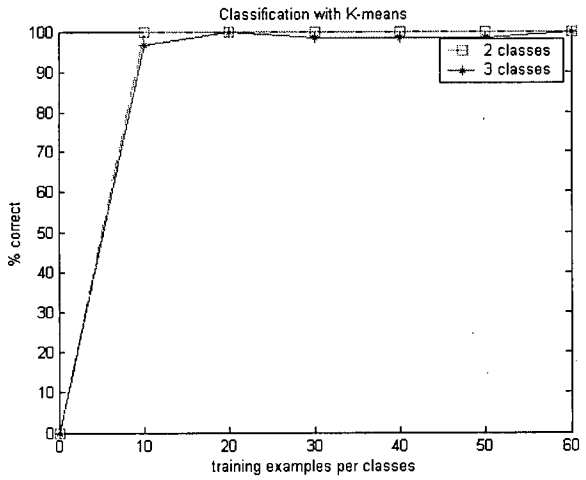

Figure 3.4: Recognition rate vs. training size for K-means method.

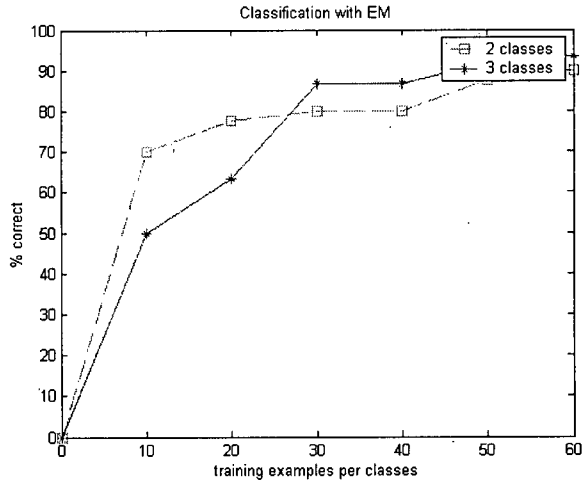

Figure 3.5: Recognition rate vs. training size for EM method.

The linear machine classifier used by Rubine that is trained with 10 or more examples can recognize 3 gesture classes with correctness $> 60\%$, and for 2 gesture classes, the adequacy is $> 85\%$. This method gives better performance when there are 40 training examples per class, but the correctness falls after increasing the number of examples.

The recognition adequacy of EM method is low when the number of training examples is less than 30 per class, however, it increases when more training examples added. The more training examples per class, the higher the correctness at recognizing gestures. The classifier trained with 30 or more examples can recognize 3 gesture classes at above 85% correctness, and for 2 gesture classes above 80% correctness. It is surprising to note that it will give better performance for 3 gesture classes than 2 gesture classes for large number of training examples.

3.5 Summary

The generation of a meaningful, extensible and effective feature set is essential. How many features are enough to describe a gesture without redundancy and what kind of features should be extracted is an important problem to be tackled. In this chapter, testing strokes comprise congruent lines and arcs, and we can see that thirteen features are quite efficient for classification. Given different gesture sets, some features may be useless and a small number of feature set may be applied to generate good results as well. In the next chapter, we want to classify strokes of different shapes from the ones we used in this chapter. We choose K-means method to use for our sketch recognition system. They are generally simpler and thus we will use a feature set with a smaller number of attributes.

Chapter 4

Sketch Recognition and 3D Model Construction

This chapter will discuss the main contributions of our work: a system for sketch recognition and 3D model construction. Sketch recognition consists of assigning feature labels to individual pen strokes, i.e., identifying the object features they corresponded to. It includes two phases: training and learning. We have chosen K-means method for stroke classification and we will show how we select features and apply this classifier to sketch recognition. Lastly, we will go through the details of 3D model construction with a generic example.

4.1 Sketch Recognition: Training

4.1.1 Shapes

In our system, each pen stroke symbolizes a single shape, such as a single curve or triangle. The advantage is that it facilitates shape recognition. The disadvantage is that the system requires the user to sketch features in a fairly specific fashion. We define a shape set for typical symbols, including curve, ellipse, triangle, square, open rectangle, as shown in Figure 4.1. We regard a line as being a kind of curve.

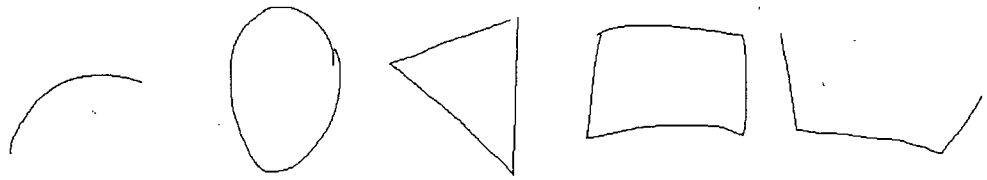

Figure 4.1: Shapes of basic symbols: arc, ellipse, triangle, square, open rectangle.

We use x and y values of the stroke points to determine its shape style, which includes two steps. For curve and open rectangle, the distance between the begin point and the end point is quite faraway; whereas for ellipse, triangle and square, the begin point is close to the end point. In this way, we can categorize the five shape types into two classes: the one with open stroke and the one with closed stroke. The next step is to employ geometrical properties of different shape type to distinguish them. Every triangle, right-angled or not, will have at least two acute angles. A square contains four right angles and an open rectangle takes two angles, whether acute or obtuse. On a stroke, the lines between a point and its two neighbouring points form an angle. This three points consist of a triangle, length of each edge can be calculated and then the cosine value of this angle. Since sketch is drawn freehand, it may incur a lot of minor noises, for instance, a line may contain some sharp angles. A good way to avoid those noises is to include more neighbouring points for calculation. With this method, angle values on a stroke can be computed one point by one point. These values change smoothly for curve and ellipse because they don't have such obvious angle traits as triangle, square and open rectangle.

4.1.2 Choice of Features

Four features as the intrinsic properties of a stroke are defined to interpret one single stroke, including:

- Feature 1 (f_1) : Normalized location in x direction.

- Feature 2 (f_2) : Normalized location in y direction.
- Feature 3 (f_3) : Drawing time.
- Feature 4 (f_4) : Drawing shape.

Once an object sketch is drawn, an axis-aligned bounding box for the whole sketch is computed. As shown in Figure 4.2, the sketch bounding box is illustrated in dotted lines. The normalized x value (feature f_1) is computed as a $f_1 = x'/x_{bb}$, where x' is the distance from the center of stroke bounding box to the left side of the sketch bounding box, and x_{bb} is the bounding box width. In the same manner, $f_2 = y'/y_{bb}$.

The drawing time (feature f_3) is measured as the total time between first and last sample point. The last feature used for stroke classification is drawing shape (feature f_4) which is categorized according to our shape set defined in Figure 4.1 and calculated from the method we discribed in 4.1.1. We assign a unique integer value to each shape type: 0 to curve, 1 to ellipse, 2 to triangle, 3 to square and 4 to open rectangle. In this way, all the features can be parameterized.

Figures 4.3 and 4.4 show the sketched components of face and rocket that will be recognized by our system. A face consists of 9 strokes drawn in any order: face, left eye, right eye, left eyebrow, right eyebrow, nose, mouth, left ear and right ear. A rocket is described by 5 strokes that can be drawn in any order: head, body, tail, left wing and right wing. Numbers in the figures correspond to the drawing order for each stroke for the example database, which provides a labelled set of training data. Detailed information is listed in Table 4.1 for face and Table 4.2 for rocket. In these tables, we also show possible drawing shapes for each component and the required shapes for some component during recognition. We set required shapes to facilitate recognition. After we aquire all the features for each stroke, we use a K-mean classifer to distinguish them.

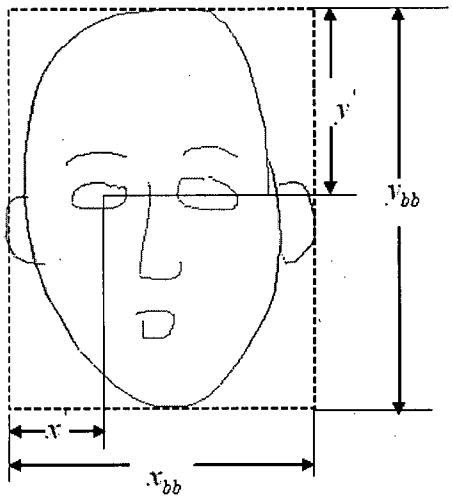

Figure 4.2: Features of head used for classification.

Order	Components	Possible Shapes	Required Shapes
1	face	ellipse	ellipse
2	left eye	ellipse	ellipse
3	right eye	ellipse	ellipse
4	left eyebrow	curve	curve
5	right eyebrow	curve	curve
6	nose	curve, triangle, ellipse	
7	mouth	curve, ellipse	
8	left ear	curve, ellipse	
9	right ear	curve, ellipse	

Table 4.1: Description of components for face.

Order	Components	Required Shapes
1	head	triangle
2	body	open rectangle
3	tail	open rectangle
4	left wing	open rectangle
5	right wing	open rectangle

Table 4.2: Description of components for rocket.

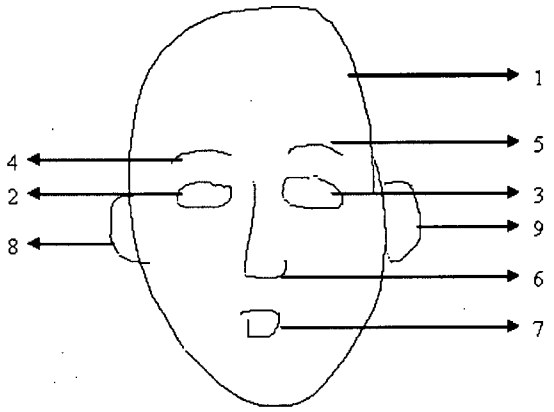

Figure 4.3: Illustration of face components.

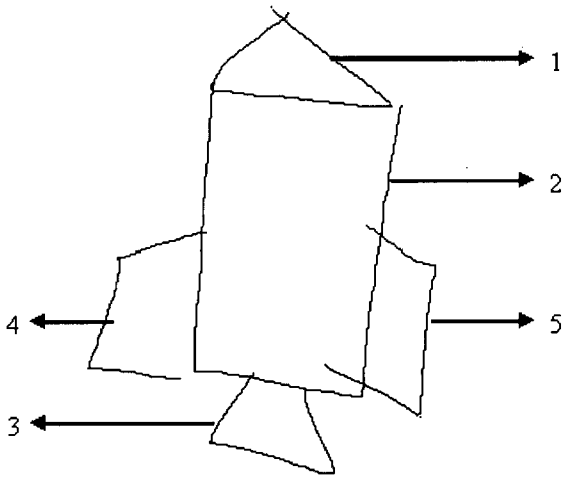

Figure 4.4: Illustration of rocket components.

4.1.3 Sketch Recognition: Matching

The gesture recognition can be regarded as classification problem for which we can apply K-means scheme as described in Chapter 3. There are C classes for classification, $C = 9$ for face and $C = 5$ for rocket. Each gestures in one class should have some similarity and can be clustered together.

4.2 3D Model Construction

In the modeling phase, the recognized strokes are used to construct 3D models. Once classified, a stroke plays a key role in determining model's geometrical construction. For our current system, curves and ellipses are rendered into ellipsoids, trianglea become cones and rectangles into cylinders.

Figure 4.5 illustrates parameters used for modeling cylinder and ellipse sketches into 3D models. There are 4 parameters for modeling a tapered cylinder: $r1$, $r2$, h and *nslices*. h represents cylinder's height, $r1$ and $r2$ are the radius of two ends of tampered cylinder. *nslices* is the the number of polygons used to represent the cylinder, which can be considered to be open on either end. If the shape of a sketch is triangle, r1 is zero. The parameters for representing ellipsoid are: r, h and *nslices*. The cross section of an ellipsoid along z axis is an ellipse with its minor axis value to be r and major axis value to be h. All the radius and height values are taken from axis-aligned bounding box around a stroke.

After sketch recognition in the previous stages, we have determined each component and its shape for an object. Because we have a priori knowledge about what kind of object is being modeled, once we identity each stroke, we can locate their position related to other components, for instance, nose and eyes are on the face while ears are at the two sides of a face.

The following is an example description of how we model a face from raw sketches. Figure 4.3 shows 9 strokes of a face. Stroke 1 is the head outline and

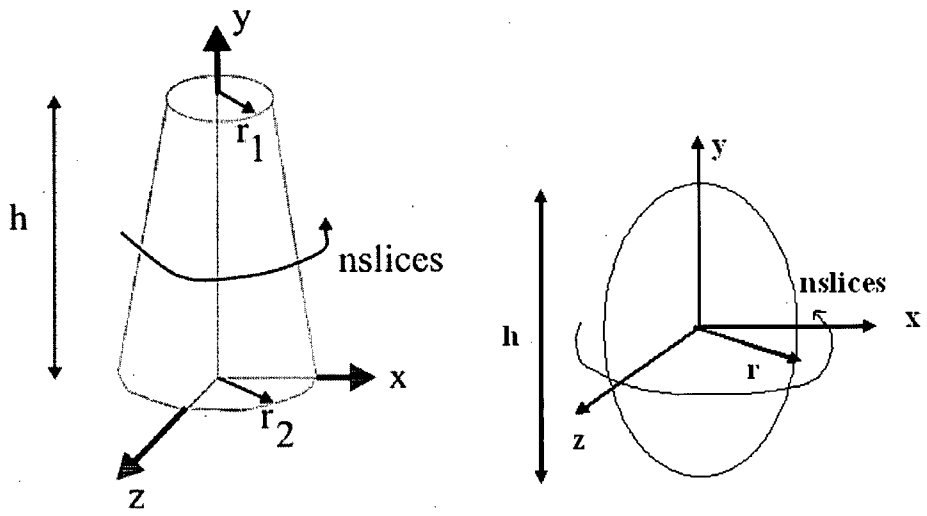

Figure 4.5: Modeling parameters for tapered cylinder and ellipsoid.

the resulting 3D head is modeled using an ellipsoid. Stroke 2 is the left eye and is modeled using an ellipsoid. Stroke 4 is left eyebrow modeled using a 3D curve. We assume eyes, eyebrows, nose and mouth are all located on the surface of the head and that the ears are located on the two sides of the head. Even if the ear strokes are not drawn in this way, we still model them as a normal face. Since they are related to head model, the z value of these models should be calculated from the model of the head outline. For the head, the cross section along the z and x axis of an ellipsoid is an ellipse and the cross section along y axis is a circle. The position values (x, y, z) of any point on an ellipsoid can be computed from ellipse and circle mathematical equations.

In the next chapter, we will show some concrete examples of human faces and rockets that go from raw sketches to 3D models.

Chapter 5

Results

We test our methods with sketch-based modeling of simple faces and rockets. Results from our system are demonstrated in this chapter. We show examples of both successful and unsuccessful sketch-based modeling. We assume that a face always takes 9 strokes and a rocket with 5 strokes, otherwise, the system will not work properly.

5.1 Face Sketching

Figure 5.1 shows three examples of faces drawn and recognized by our system. We show examples of both successful and unsuccessful sketch-based modeling. From left to right are the sketches, front view of the 3D models and a rotated view of the 3D models.

Figure 5.3 shows two faces drawn in our system with incorrect recognition. Table 5.1 list the drawing components and recognized ones for these two examples. In example one, "mouth" is mistaken as a "left ear" because these two strokes are drawn in the same shape and other features data are similar and therefore confuse the classifier. In example two, "left eye" is wrongly recognized as "left eyebrow" because they are drawn too close to each other. One possible solution is to make each component unique during recognition. Each component can be marked and if

Drawing Order	Example 1		Example 2	
	Drawing Components	Recognized Components	Drawing Components	Recognized Components
1	face	face	left eyebrow	left eyebrow
2	left ear	left ear	right eyebrow	right eyebrow
3	right ear	right ear	left eye	left eye
4	left eyebrow	left eyebrow	right eye	right eyebrow
5	left eye	left eye	nose	nose
6	right eyebrow	right eyebrow	mouth	mouth
7	right eye	right eye	face	face
8	nose	nose	left ear	left ear
9	mouth	left ear	right ear	right ear

Table 5.1: Comparisons of drawing and recognized face components with incorrect recognition.

there is another component also marked as "left eyebrow", we can choose the label with the least error according to that classifier.

5.2 Rocket Sketching

Three examples of rockets are shown in Figure 5.2. From left to right are the raw sketches, front image of rendered rockets and rocket images after rotation.

5.3 Discussion

We set up a training library for sketches including face and rocket, the size of database will affect recognition correctness. From Chapter 3, we know that the more examples the database contains, the more accurately the sketches will be recognized. Our training data consists of 40 face skteches and 30 rocket sketches. Classifiers are obtained after applying K-means clustering to this data.

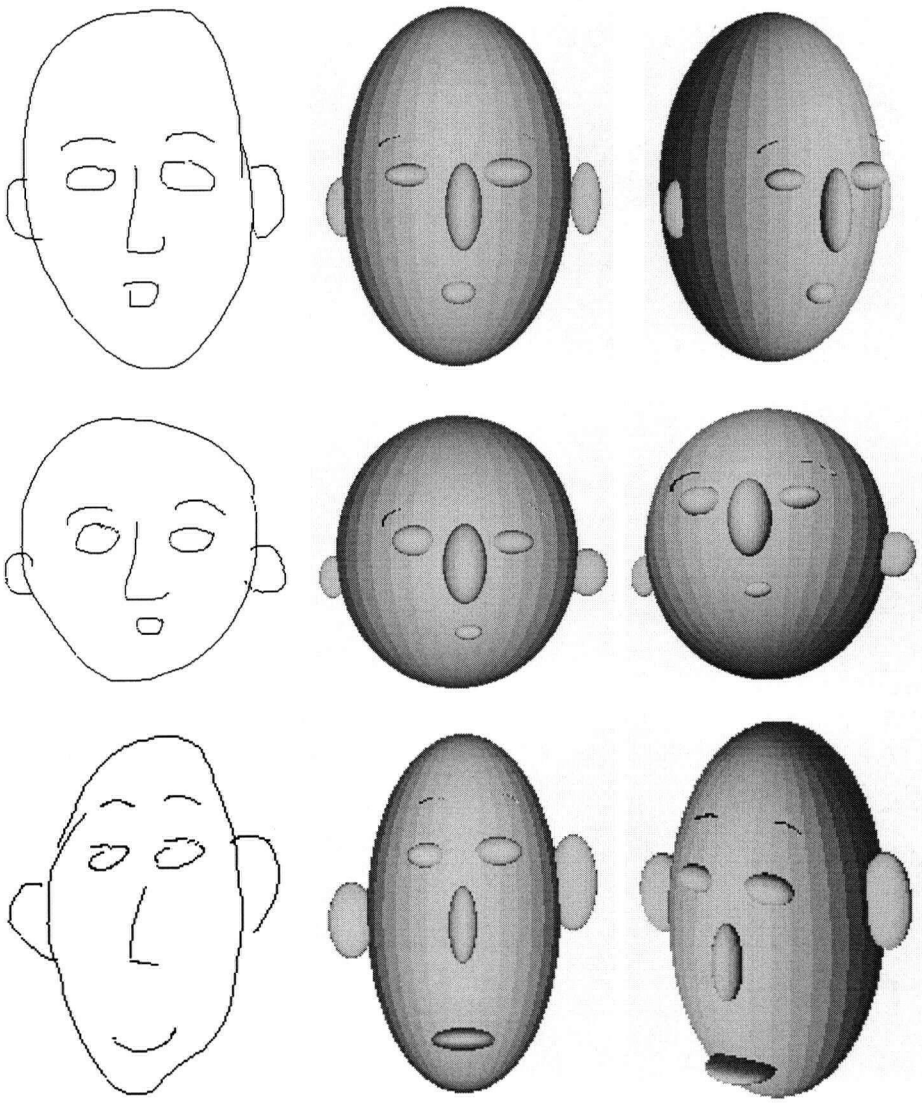

Figure 5.1: Face modeling with correct recognition.

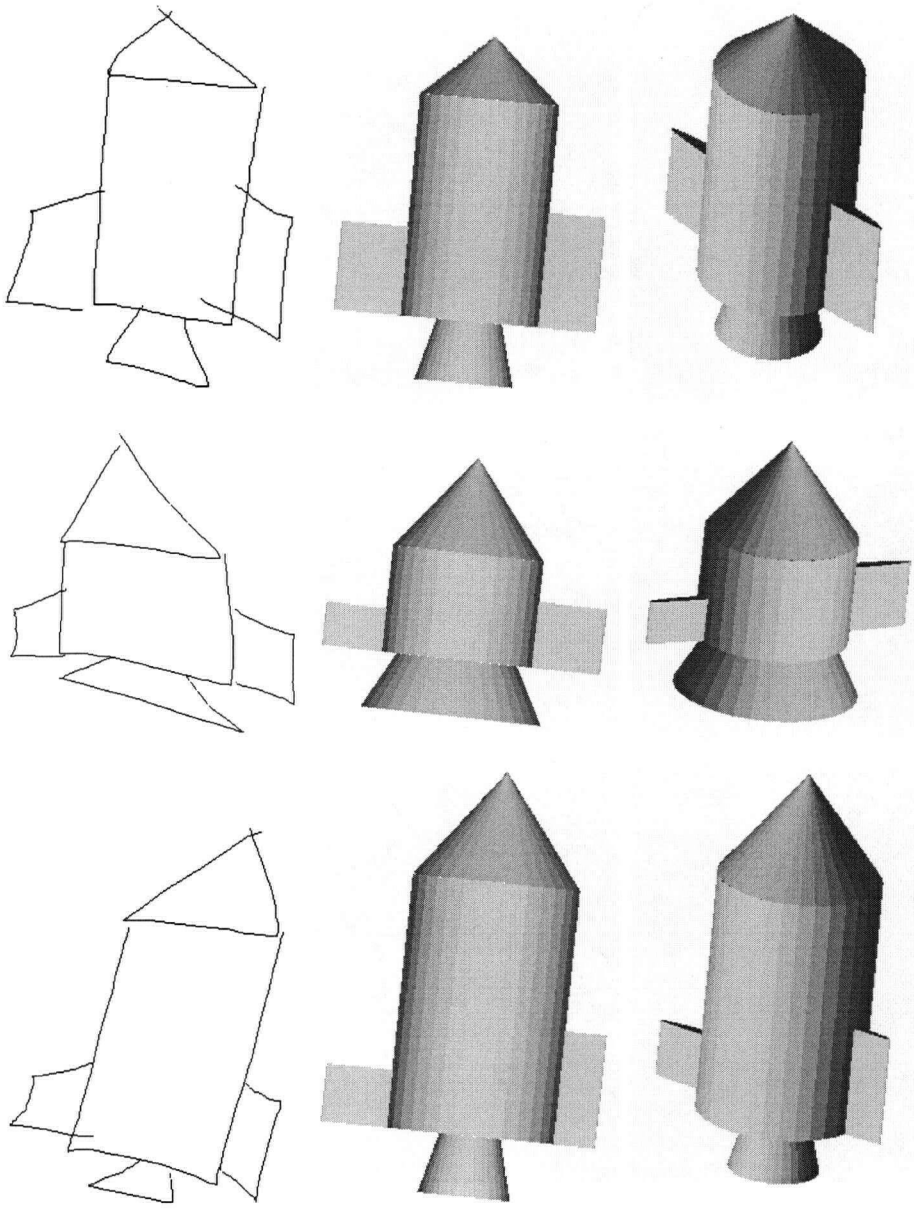

Figure 5.2: Rocket modeling with correct recognition.

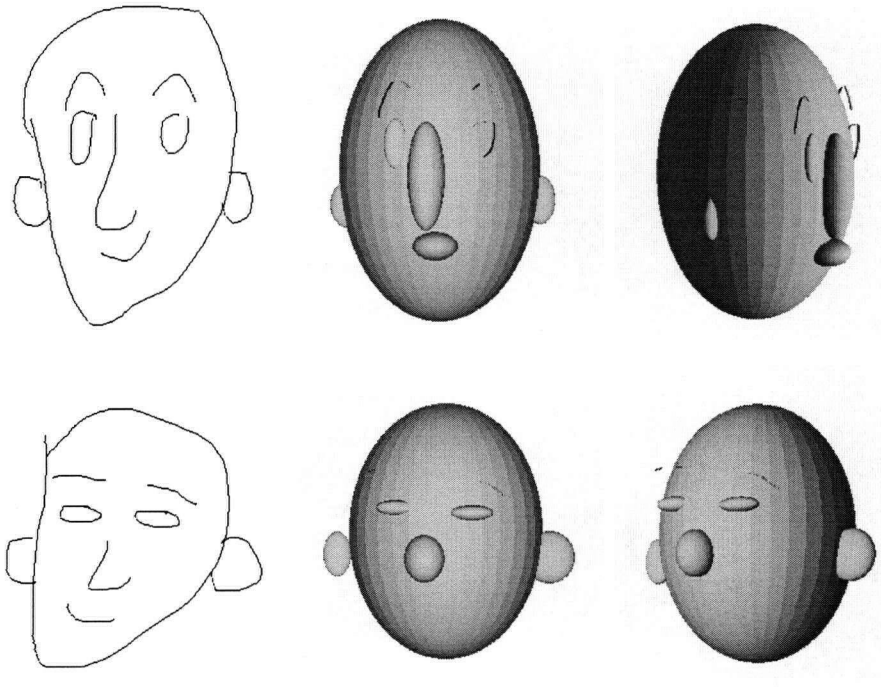

Figure 5.3: Face modeling with incorrect recognition.

Chapter 6

Conclusions and Future Work

The system has been tested on simple drawings such as human faces and rockets. In order to make the system work for a new class of objects, we have to specify how many components this object has and from some training examples, collect the shapes that can best symbolize this component. With this information, the system can create 3D models from simple sketches.

Our current algorithms and implementation are efficient enough for experimental use. However, they can fail or generate unintuitive results when the user draws unexpected strokes. We must devise more robust and flexible algorithms to handle a variety of user inputs. In particular, we plan to enhance the algorithm to allow multi-stroke input. Currently, we assume that each stroke represent one shape. If we would allow pen strokes to represent any number of shape primitives connected together, the system could potentially cope deal with more complicated sketches.

Another important research direction is to develop additional features to support a wider variety of shapes with arbitrary topology, and to allow more precise control of the shape. Second, in training phase, we would like to test with more classifiers to construct a more accurate classifier for matching strokes.

Bibliography

[1] James Arvo and Kevin Novins. Fluid sketches: Continuous recognition and morphing of simple hand-drawn shapes. In *Proceedings of the 13th Annual ACM Symposium on User Interface Software and Technlogy*, November 2000.

[2] Berge Belongie, Jitendra Malik, and Jan Puzicha. Shape matching and object recognition using shape contexts. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 24(24):509–522, 2002.

[3] Chris Calhoun, Thomas F. Stahovich, Tolga Kurtoglu, and Levent Burak Kara. Recognizing multi-stroke symbols. *AAAI Spring Symposium on Sketch Understanding*, pages 15–23, 2002.

[4] F Cohen, Z Huang, and Z Yang. Invariant matching and identification of curves using b-splines curve representation. *IEEE Transactions on Image Processing*, 4(1):1–10, 1995.

[5] M. Gross and E Do. Ambiguous intentions: a paper-like interface for creative design. In *Proceedings of UIST 96*, pages 183–192, 1996.

[6] T Igarashi, S Matsuoka, S Kawachiya, and H Tanaka. Interactive beautification: A technique for rapid geometric design. In *UIST'97*, pages 105–114, 1997.

[7] Takeo Igarashi and John F. Hughes. A suggestive interface for 3d drawing. *ACM UIST Symposium on User Interface Software and Technology*, pages 173–181, 2001.

[8] Takeo Igarashi, Satoshi Matsuoka, and Hidehiko Tanaka. Teddy: A sketching interface for 3d freefrom design. *SIGGRAPH 99, Computer Graphics Proceedings*, pages 409–416, 1999.

[9] Michael I. Jordan. *An Introduction to Probabilistic Graphical Models.* To be published, 2004.

[10] T. D. Kimura, A Apte, and S. Sengupta. A graphic diagram editor for pen computers. *Software Concepts and Tools*, pages 82–95, 1994.

[11] D. Lamb and A Bandopadhay. Interpreting a 3d object from a rough 2d line drawing. In *Proc First IEEE Conf on Visualization, 90*, pages 50–66, 1990.

[12] James A. Landay and Brad A. Myers. Sketching interfaces: Toward more human interface design. *IEEE Computer*, 34(3):56–64, 2001.

[13] Han Li. Freehand sketch recognition system. http://science.unitn.it/ tomasi/think/pdf/pha.ppt.

[14] Hod Lipson and Moshe Shpitalni. Correlation-based reconstruction of a 3d object from a single freehand sketch. *2002 AAAI Spring Symposium on Sketch Understanding*, pages 99–104, 2002.

[15] Brian T. Luke. K-means clustering. http://fconyx.ncifcrf.gov/ lukeb/kmeans.html.

[16] P. Sinha E. Osuna M. Oren, C. Papageorgiou and T. Poggio. Pedestrian detection using wavelet templatess. In *Proc. IEEE Conf. Comput. Vision and Pattern Recognition*, pages 193–199, June 1997.

[17] James V. Mahoney and Markus P. J. Frommerz. Three main concerns in sketch recognition and an approach to addressing the. *Sketch Understanding, Papers from the 2002 AAAI Spring Symposium*, pages 105–112, 2002.

[18] Zhao R. Incremental recognition in gesture-based and syntax directed diagram editor. In *Proceedings of InterCHI'93*, pages 95–100, 1993.

[19] Dean Rubine. Integrating gesture recognition and direct manipulation. In *Proceedings of the Summer'91 USENIX Technical Conference*, pages 95–100, 1991.

[20] Dean Rubine. Specifying gestures by example. *SIGGRAPH 91, Computer Graphics Proceedings*, 25:329–337, 1991.

[21] H. Schneiderman and T. Kanade. A statistical method for 3 dimensional object detection applied to faces and cars. In *Proc. IEEE Conf. Comput. Vision and Pattern Recognition*, 2000.

[22] Moshe Shpitalni and Hod Lipson. Classification of sketch strokes and corner detection using conic sections and adaptive clustering. *ASME Journal of Mechanical Design*, 119(2):131–135, 1997.

[23] T. F Stahovich. Sketchit: a sketch interpretation tool for conceptual mechanism design. *Technical report, MIT AI Laboratory*, 1996.

[24] T. F. Stahovich, R. Davis, and H Shrobe. Generating multiple new designs from a sketch. *Artificial Intelligence*, 104(1-2):211–264, 1998.

[25] M. Turk and A. Pentland. Eigenfaces for recognition. *J. Cognitive Neuroscience*, 3(1):71–96, 1991.

[26] P. Viola and M. Jones. Robust real-time object detection. *Second international workshop on statistical and computational theories of vision*, 2001.

[27] L. Weisman. A foundatoin for intelligent multimodal drawing and sketching programs. 1999.

[28] D. Geman Y. Amit and K. Wilder. Joint induction of shape features and tree classifiers. *IEEE Trans. Pattern Analysis and Machine Intelligence*, 19(11):1300–130596, November 1997.

[29] Bo Yu and Shijie Cai. A domain-independent system for sketch recognition. 2003.

[30] Robert C. Zeleznik, Kenneth P. Herndon, and John F. Hughes. Sketch: An interface for sketching 3d scenes. *SIGGRAPH 96, Computer Graphics Proceedings*, pages 163–170, 1996.

[31] Hao Zhang and Jitendra Malik. Learning a discriminative classifier using shape context distances. *IEEE Computer Vision and Pattern Recognition*, pages 242–247, 2003.

Appendix A

User Interface

The physical user interface of this sketch system is implemented with FLTK, a platform independent graphics toolkit. The interface consists of a main display window on the left side and a control panel on the right. Most control actions are focused on drawing gestures and displaying object, we use a two-button mouse for drawing and control. The top panel called "Object Display" contain a number of buttons, the function of which is given below.

- sketch: ready for new sketching the object or add more sketches to an rendered object.
- solid shading: show object with solid shading.
- render: render the whole sketches after finished drawing an object.
- show normal: display normal on each point of object meshes.
- reset: clean the display window and ready to draw other objects.
- redisplay: show original sketches.

In the middle, there is a animation panel designed for showing some simple animations. Different training and drawing choice buttons are listed at the buttom

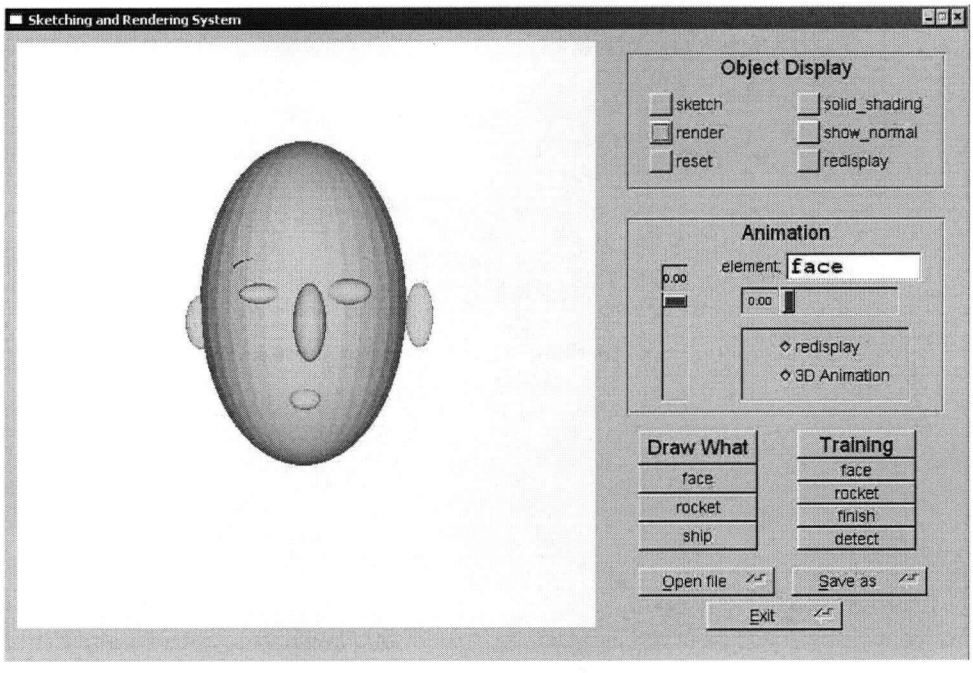

Figure A.1: The interface of modeling system.

of the control panel. In addition to gestures, we also use a few button widgets for auxiliary operations, such as save and load.